3. 신경망 시작하기

- 단일층 신경망 구현을 이해한다.
- 검증 세트 분리와 스케일링을 이해한다.
- 과대적합과 과소적합을 이해한다.
- 규제방법을 배우고 신경망에 적용한다.
- 교차 검증을 이해하고 구현한다.

3. 신경망 시작하기

3.1 단일층 신경망 구현

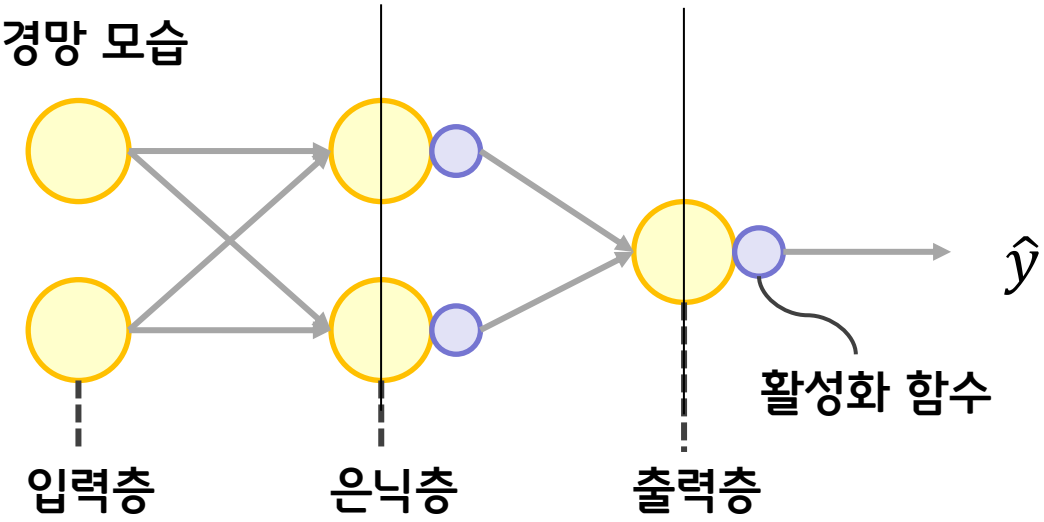
3.2 검증 세트 분리와 스케일링

3.3 과대적합과 과소적합

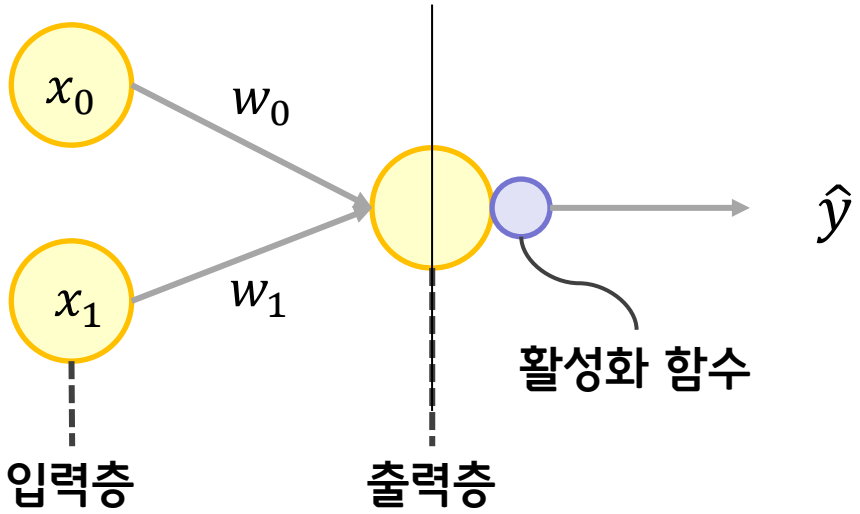
3.4 규제방법 구현

3.5 교차 검증 구현

일반적인 신경망 모습



단일층 신경망

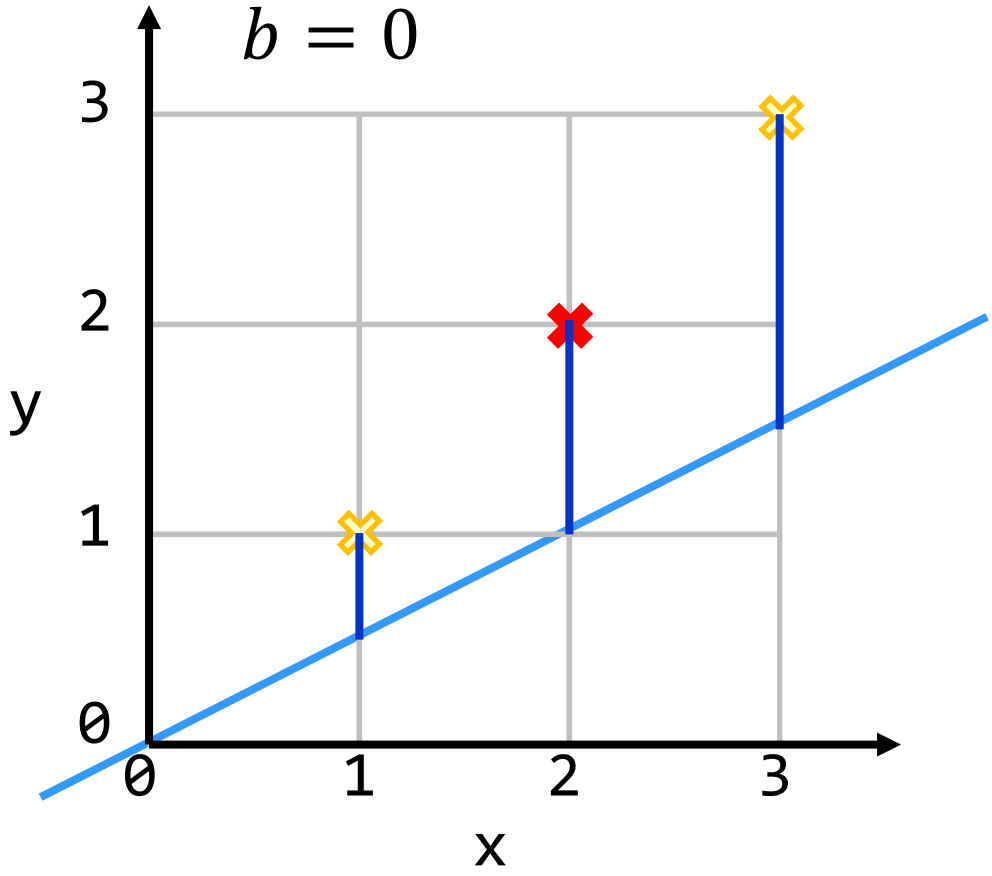


손실 함수의 결과값 조정해 저장 기능 추가하기

```
def __init__(self):
    self.losses = []        # 손실 함수의 결과값을 저장할 리스트

def fit(self, x, y, epochs=100):
        ...
          for i in indexes:                          # 모든 샘플에 대해 반복합니다
              z = self.forpass(x[i])                 # 정방향 계산
              a = self.activation(z)                 # 활성화 함수 적용
              err = -(y[i] - a)                      # 오차 계산
              w_grad, b_grad = self.backprop(x[i], err) # 역방향 계산
              self.w -= w_grad                       # 가중치 업데이트
              self.b -= b_grad                       # 절편 업데이트
              # 안전한 로그 계산을 위해 클리핑한 후 손실을 누적합니다
              a = np.clip(a, 1e-10, 1-1e-10)
              loss += -(y[i]*np.log(a)+(1-y[i])*np.log(1-a))
          # 에포크마다 평균 손실을 저장합니다
          self.losses.append(loss/len(y))
```

a가 0에 가까워지면 np.log() 함수의 값은 음의 무한대가 되고 a가 1에 가까워지면 np.log() 함수의 값은 0이 된다. 손실값이 무한해 지면 정확한 값을 계산할 수 없으므로 $-1 * 10^{-10} \sim 1 - 1 * 10^{10}$ 사이가 되록 np.clip() 함수로 조정한다.

$$\hat{y} = wx + b$$
$$w = 0.5$$
$$b = 0$$



$\frac{1}{2}(\hat{y} - y)^2$ 확률적 경사하강

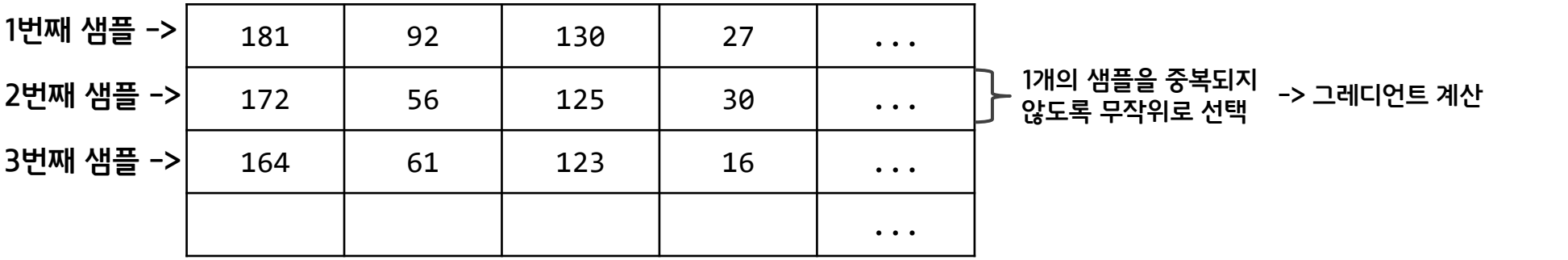
미니배치 경사하강

$\frac{1}{2m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$ mse

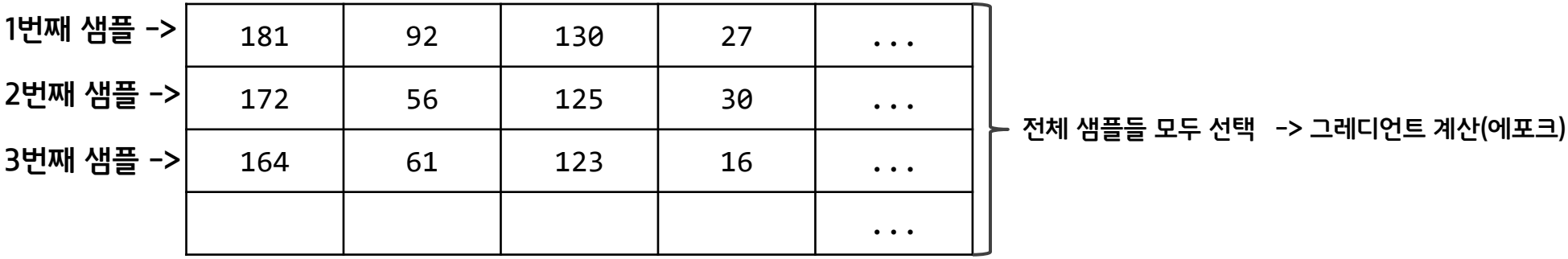
배치 경사하강

(0.25+1+2.25)

여러 가지 경사 하강법



확률적 경사 하강법



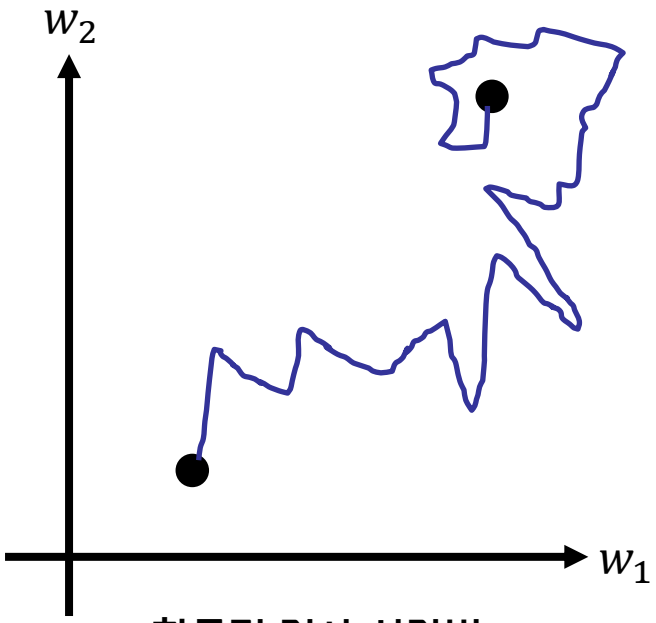
배치 경사 하강법

여러 가지 경사 하강법

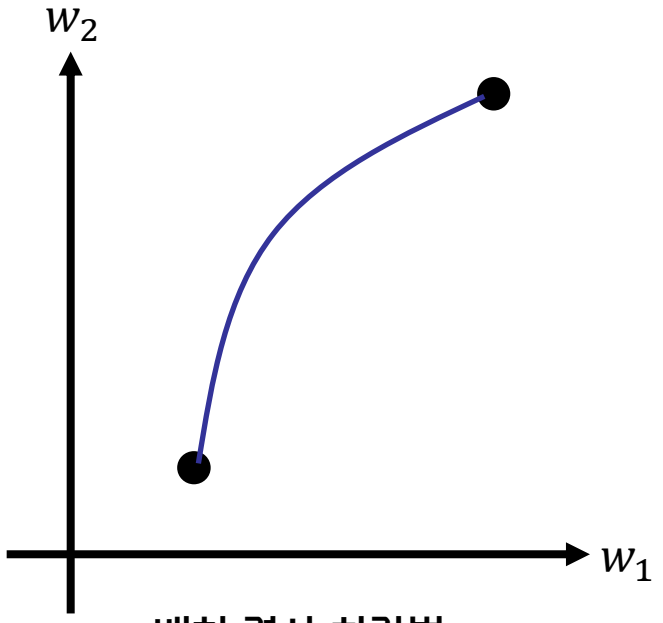
1번째 샘플 ->	181	92	130	27	...
2번째 샘플 ->	172	56	125	30	...
3번째 샘플 ->	164	61	123	16	...
					...

전체 샘플 중 몇 개의
샘플을 중복되지 않도록 -> 그레디언트 계산
무작위로 선택

미니 배치 경사 하강법



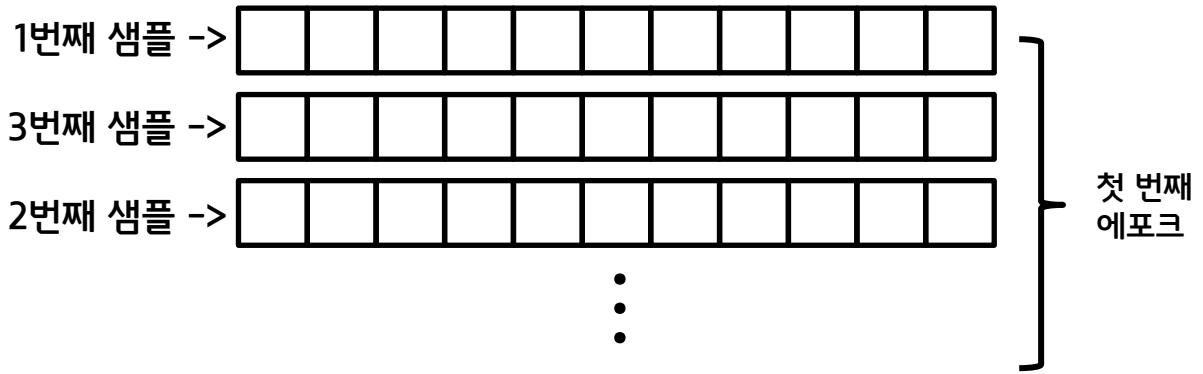
확률적 경사 하강법

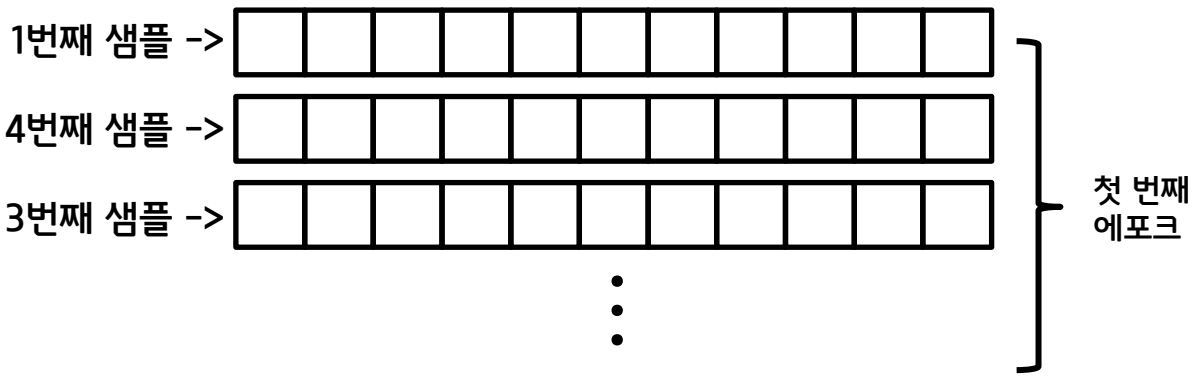


배치 경사 하강법

매 에포크 마다 훈련 세트의 샘플 순서 섞기

SGD : 확률적 경사하강





`np.random.permutation()` **함수를 사용하여 인덱스를 섞을수 있다.**

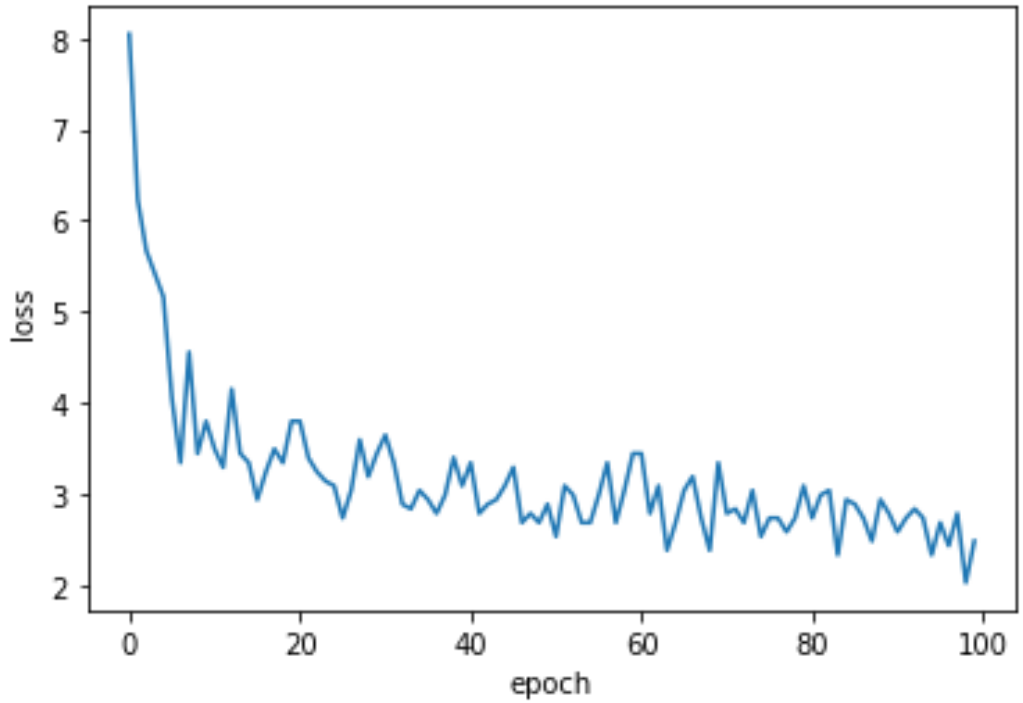
평균 손실 저장 후 시각화

```
def fit(self, x, y, epochs=100):
    self.w = np.ones(x.shape[1])           # 가중치를 초기화합니다.
    self.b = 0                             # 절편을 초기화합니다.
    for i in range(epochs):                # epochs만큼 반복합니다
        loss = 0
        indexes = np.random.permutation(np.arange(len(x)))
        for i in indexes:
            ...
            loss += -(y[i]*np.log(a)+(1-y[i])*np.log(1-a))
            # 에포크마다 평균 손실을 저장합니다
            self.losses.append(loss/len(y))
    ...

layer = SingleLayer()
layer.fit(x_train, y_train)
layer.score(x_test, y_test)
```

`loss += -(y[i]*np.log(a)+(1-y[i])*np.log(1-a))` **로 손실 함수의 결과 값을 저장 후**
`self.losses.append(loss/len(y))` **로 평균 손실을 저장한 후**
`layer.score(x_test, y_test)` **로 시각화 하여 출력한다.**

평균 손실 저장 후 시각화

```
plt.plot(layer.losses)
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```



3. 신경망 시작하기

3.1 단일층 신경망 구현

3.2 검증 세트 분리와 스케일링

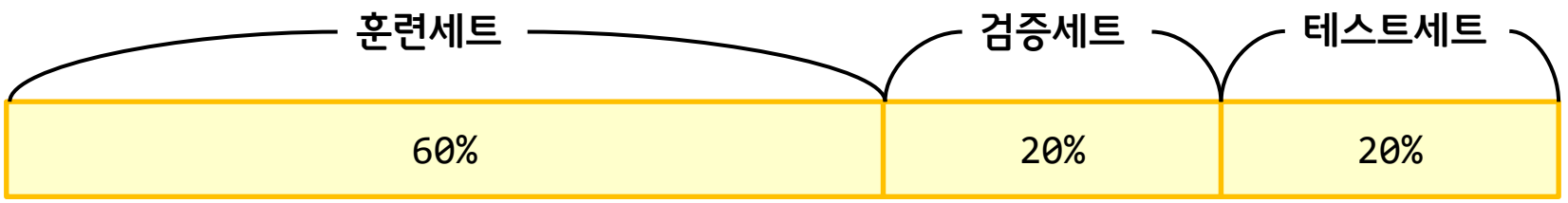
3.3 과대적합과 과소적합

3.4 규제방법 구현

3.5 교차 검증 구현

"테스트 세트로 모델를 튜닝하면 실전에서 좋은 성능을 기대하기 어렵다"

검증 세트 분리




```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
cancer = load_breast_cancer()
x = cancer.data
y = cancer.target
x_train_all, x_test, y_train_all, y_test = train_test_split(x, y, stratify=y,
test_size=0.2, random_state=42)

x_train, x_val, y_train, y_val = train_test_split(x_train_all, y_train_all,
stratify=y_train_all, test_size=0.2, random_state=42)
print(len(x_train), len(x_val))
```

```
364 91
```

데이터 전처리와 특성의 스케일

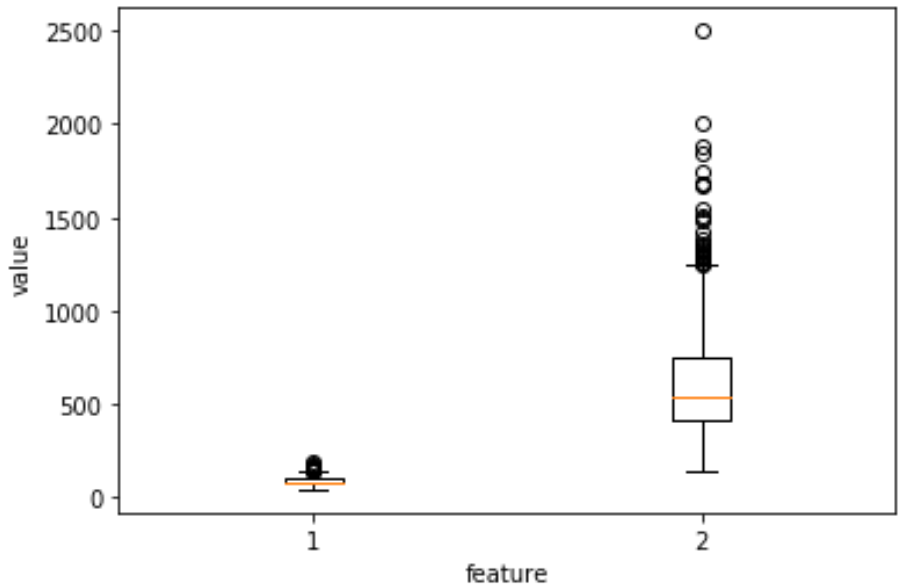
	당도	무게	...
사과1	4	540	...
사과2	8	700	...
사과3	2	480	...

사과의 당도는 1~10이고 사과의 무게의 범위는 500~1000이다.
이런 경우 '두 특성의 스케일 차이가 크다'라고 말한다.

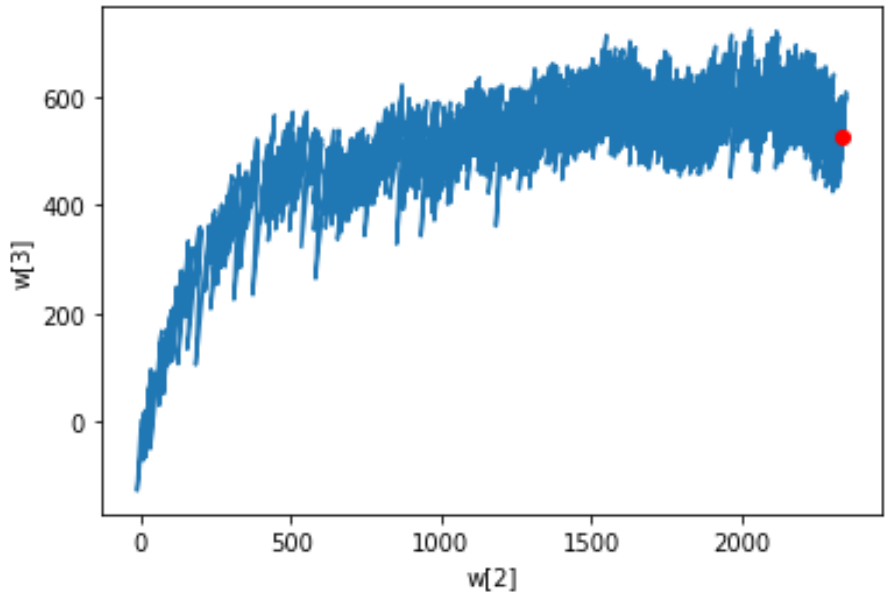
데이터 전처리와 특성의 스케일

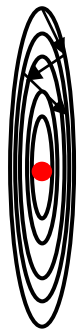
```
print(cancer.feature_names[[2,3]])
plt.boxplot(x_train[:, 2:4])
plt.xlabel('feature')
plt.ylabel('value')
plt.show()
```

['mean perimeter' 'mean area']



```
w2 = []
w3 = []
for w in layer1.w_history:
    w2.append(w[2])
    w3.append(w[3])
plt.plot(w2, w3)
plt.plot(w2[-1], w3[-1], 'ro')
plt.xlabel('w[2]')
plt.ylabel('w[3]')
plt.show()
```





스케일 조정

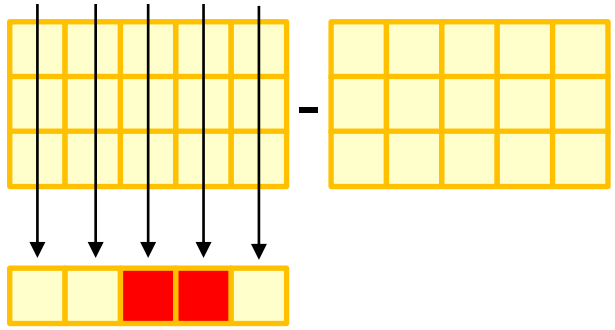
$$z = \frac{x - \mu}{s}$$

표준화는 특성 값에서 평균을 빼고
표준 편차로 나누면 된다.

$$s = \sqrt{\frac{1}{m}\sum_{i=0}^{m}(x_i - \mu)^2}$$

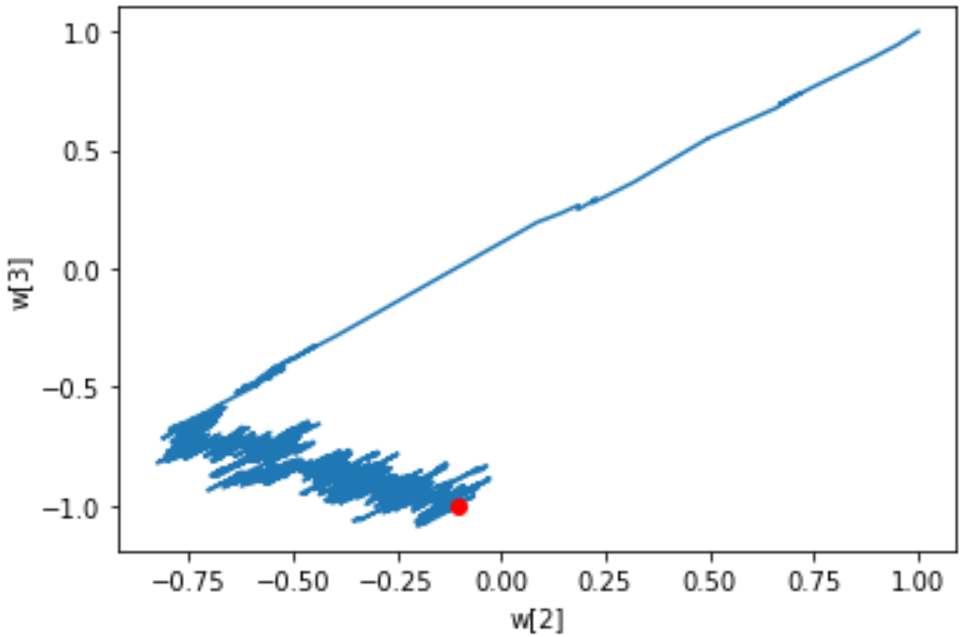
표준 편차 공식

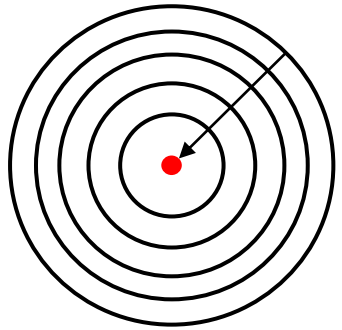
(3, 5)



```
train_mean = np.mean(x_train, axis=0)
train_std = np.std(x_train, axis=0)
x_train_scaled = (x_train - train_mean) / train_std
```

```
w2 = []
w3 = []
for w in layer2.w_history:
    w2.append(w[2])
    w3.append(w[3])
plt.plot(w2, w3)
plt.plot(w2[-1], w3[-1], 'ro')
plt.xlabel('w[2]')
plt.ylabel('w[3]')
plt.show()
```





```
layer2.score(x_val, y_val)
```

0.37362637362637363

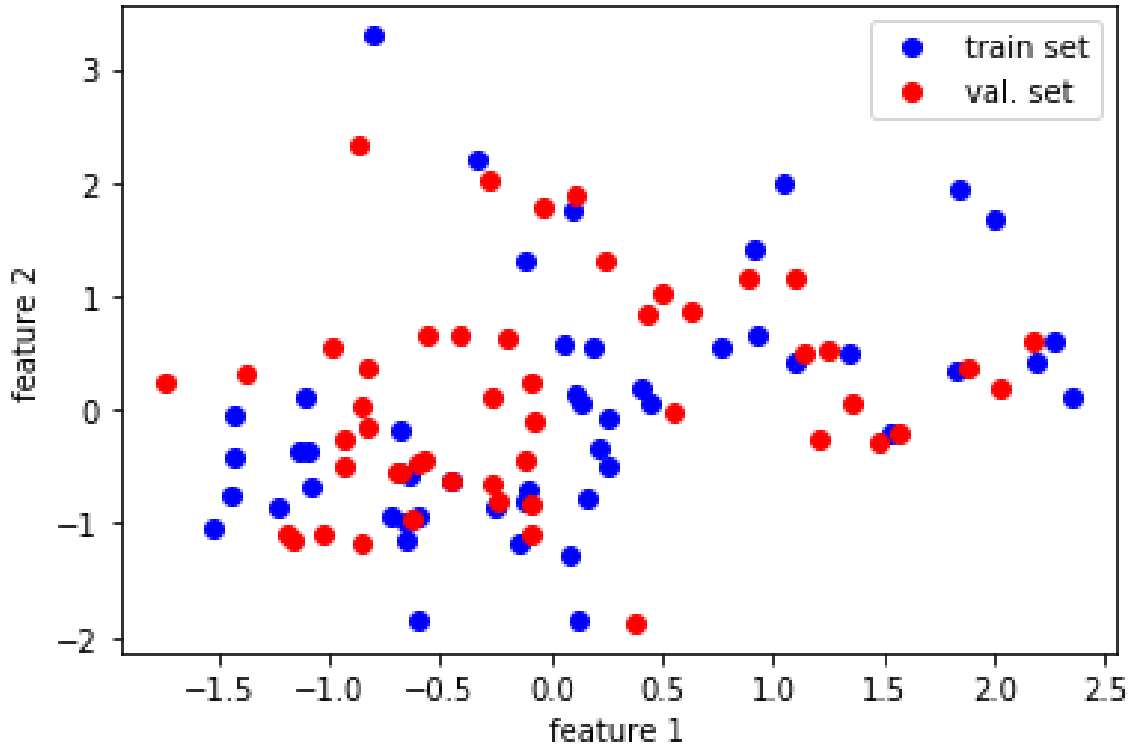
훈련 세트만 스케일을 조정하면 성능이 좋지않다.

검증세트도 표준화 전처리를 적용해야 한다.

```
val_mean = np.mean(x_val, axis=0)
val_std = np.std(x_val, axis=0)
x_val_scaled = (x_val - val_mean) / val_std
layer2.score(x_val_scaled, y_val)
```

0.967032967032967

```
plt.plot(x_train_scaled[:50, 0], x_train_scaled[:50, 1], 'bo')
plt.plot(x_val_scaled[:50, 0], x_val_scaled[:50, 1], 'ro')
plt.xlabel('feature 1')
plt.ylabel('feature 2')
plt.legend(['train set', 'val. set'])
plt.show()
```



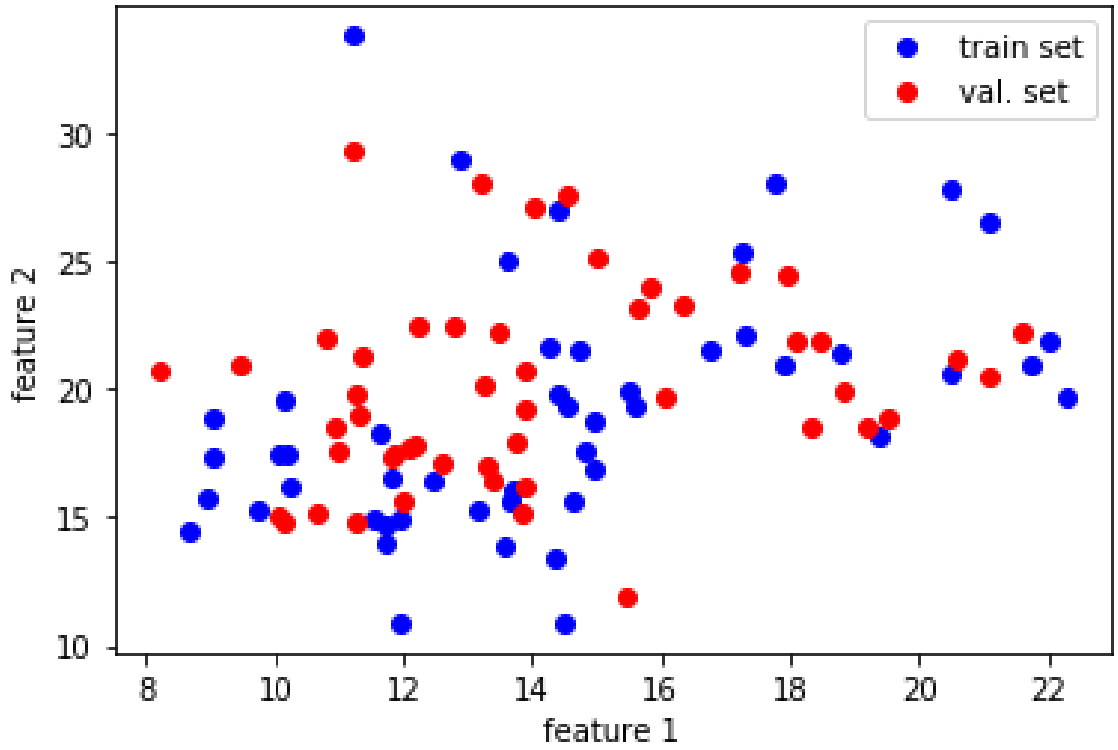
훈련 세트와 검증 세트의 거리가 그대로 유지 되어야 한다.

하지만 지금은 거리가 달라졌다.

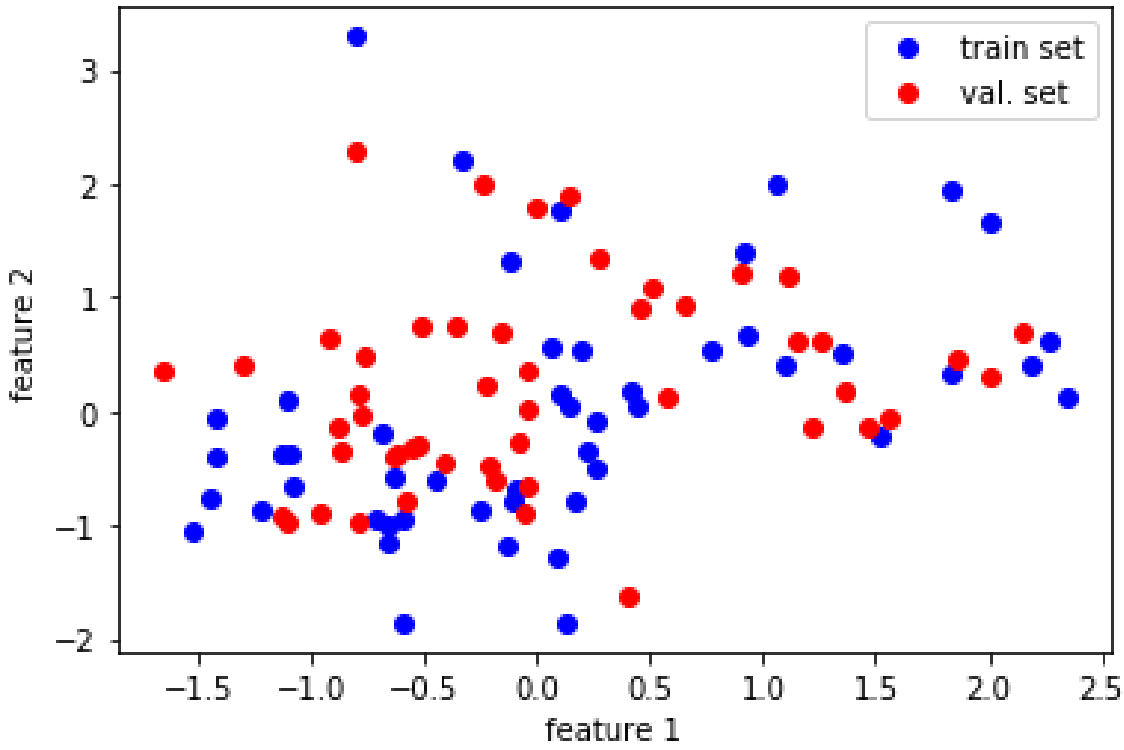
이유는

훈련 세트과 검증 세트가 각각 다른 비율로 전처리 되었기 때문이다.

```
plt.plot(x_train[:50, 0], x_train[:50, 1], 'bo')
plt.plot(x_val[:50, 0], x_val[:50, 1], 'ro')
plt.xlabel('feature 1')
plt.ylabel('feature 2')
plt.legend(['train set', 'val. set'])
plt.show()
```



```
x_val_scaled = (x_val - train_mean) / train_std
plt.plot(x_train_scaled[:50, 0], x_train_scaled[:50, 1], 'bo')
plt.plot(x_val_scaled[:50, 0], x_val_scaled[:50, 1], 'ro')
plt.xlabel('feature 1')
plt.ylabel('feature 2')
plt.legend(['train set', 'val. set'])
plt.show()
```



훈련 세트의 평균,
표준 편차를 이용하여
검증 세트를 변환
하면
문제가 해결된다.

3. 신경망 시작하기

3.1 단일층 신경망 구현

3.2 검증 세트 분리와 스케일링

3.3 과대적합과 과소적합

3.4 규제방법 구현

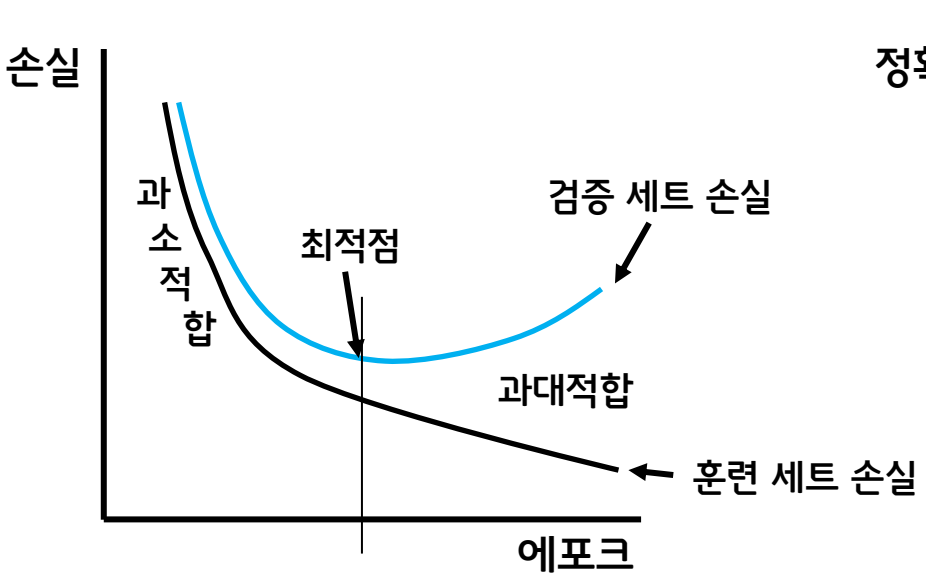
3.5 교차 검증 구현

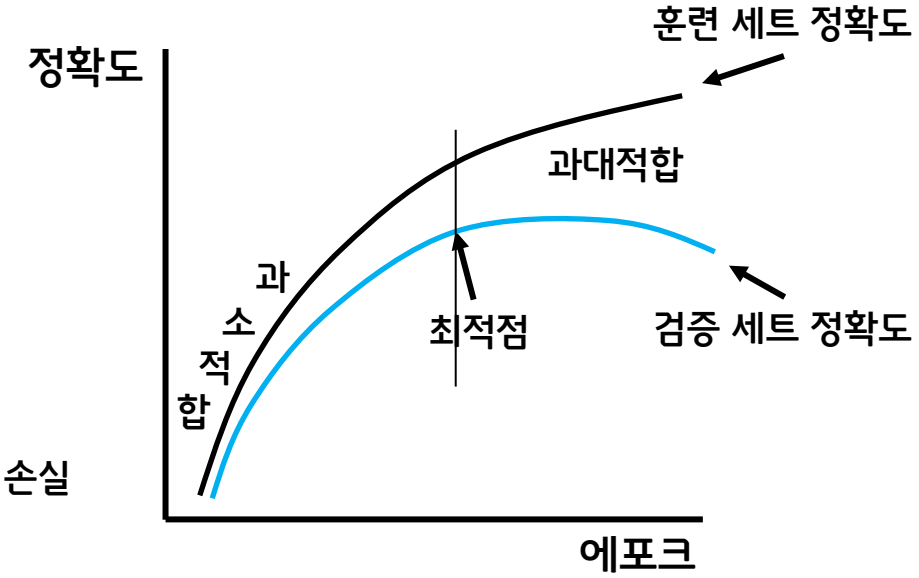
과대적합 :

훈련 세트에서는 좋은 성능을 내지만 검증 세트에서는 낮은 성능을 내는 경우

과소적합 :

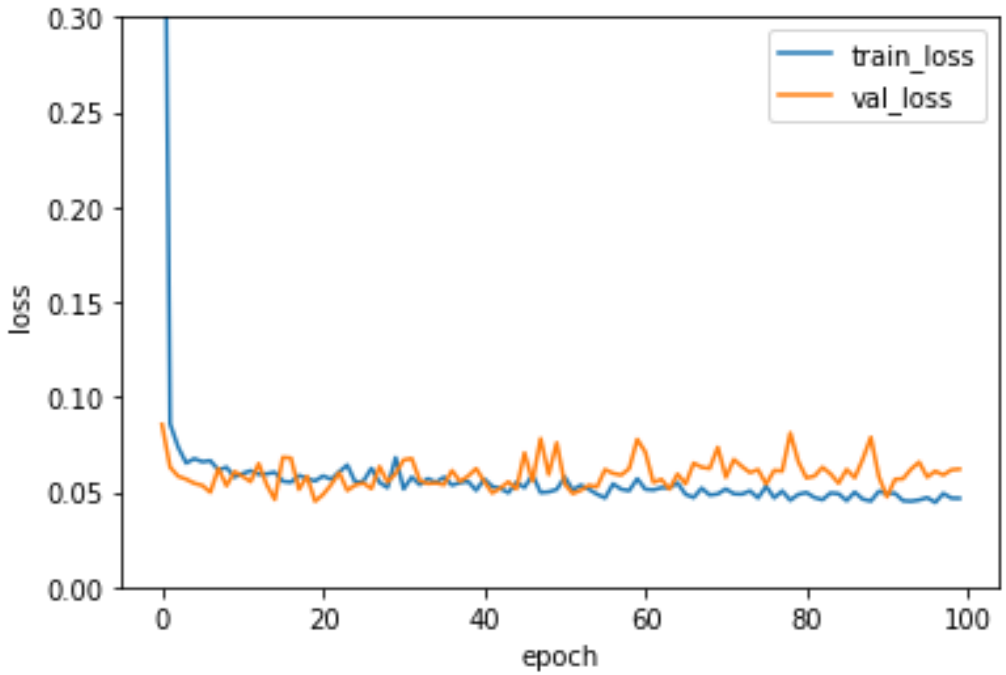
훈련 세트와 검증세트의 성능에는 차이가 크지 않지만 모두 낮은 성능을 내는 경우





과대적합 구간

```
layer3 = SingleLayer()
layer3.fit(x_train_scaled, y_train, x_val=x_val_scaled, y_val=y_val)
plt.ylim(0, 0.3)
plt.plot(layer3.losses)
plt.plot(layer3.val_losses)
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train_loss', 'val_loss'])
plt.show()
```



훈련 조기 종료

```
layer4 = SingleLayer()
layer4.fit(x_train_scaled, y_train, epochs=20)
layer4.score(x_val_scaled, y_val)
```

0.978021978021978

3. 신경망 시작하기

3.1 단일층 신경망 구현

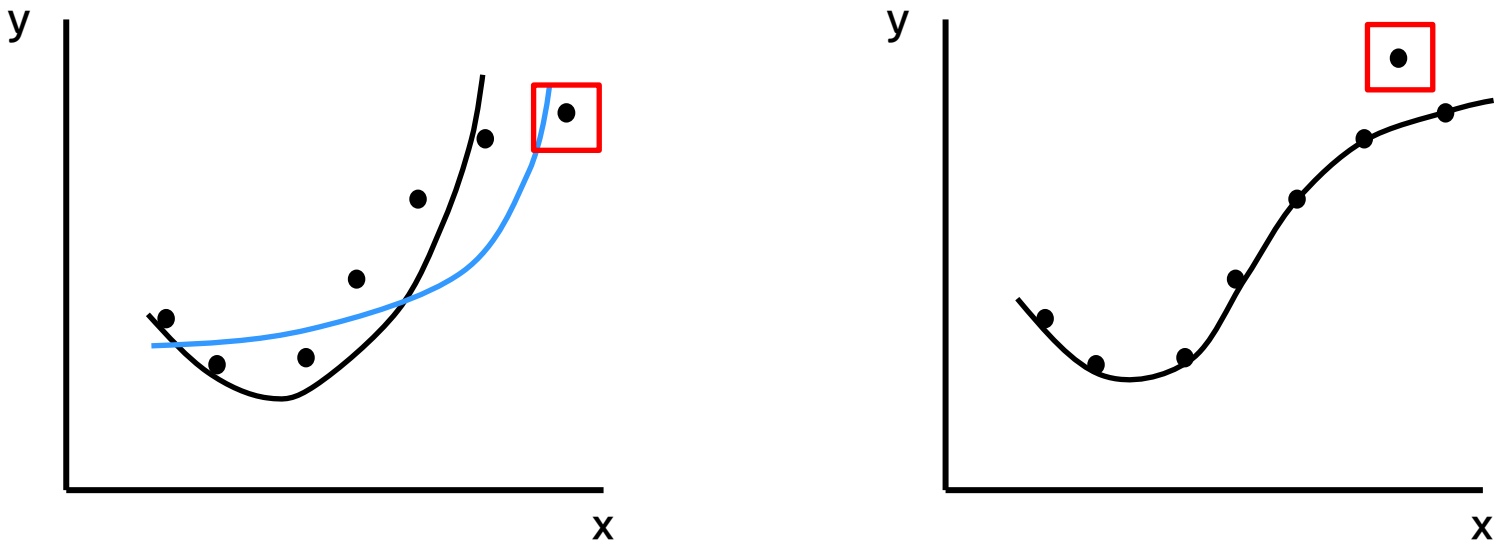
3.2 검증 세트 분리와 스케일링

3.3 과대적합과 과소적합

3.4 규제방법 구현

3.5 교차 검증 구현

가중치 규제(regularization)



L1 규제는 손실 함수에 가중치의 절대값인 L1 노름(norm)을 추가한다.

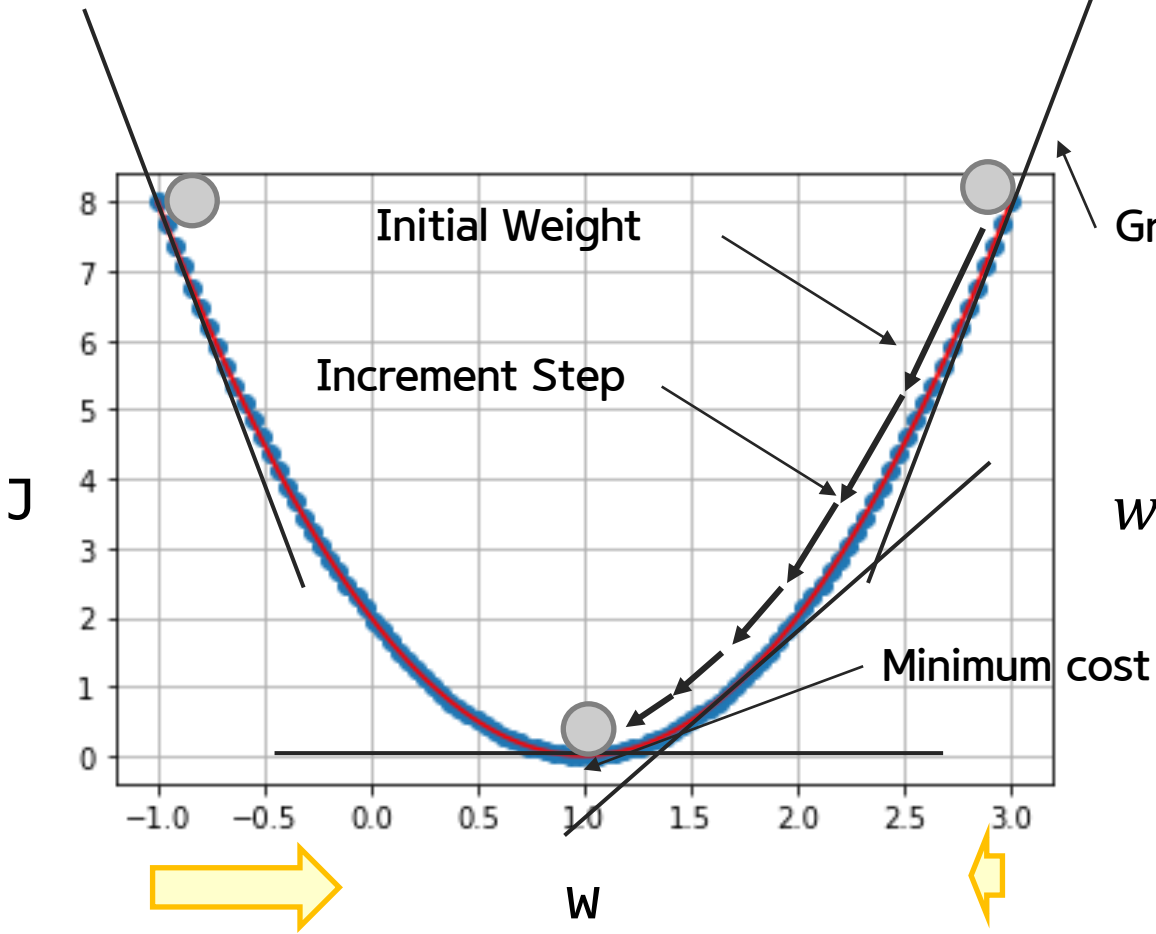
$$\left|\left|w\right|\right|_1 = \sum_{i=1}^{n} |w_i|$$

$$L = -(ylog(a) + (1-y)\log(1-a))$$

L1 노름을 그냥 더하지 않고 규제의 양을 조절하는 파라미터 α 를 곱한 후 더한다.

$$L = -(ylog(a) + (1-y)\log(1-a)) + \alpha \sum_{i=1}^{n} |w_i|$$

Gradient descent algorithm



$$\hat{y} = wx + b$$

$$w = w - \alpha(\hat{y} - y)\mathrm{x}$$

0.08

$$w = w - \alpha(\hat{y} - y)\mathrm{x} + 0.01 * \mathrm{sign(w)}$$

-0.08 + 0.01

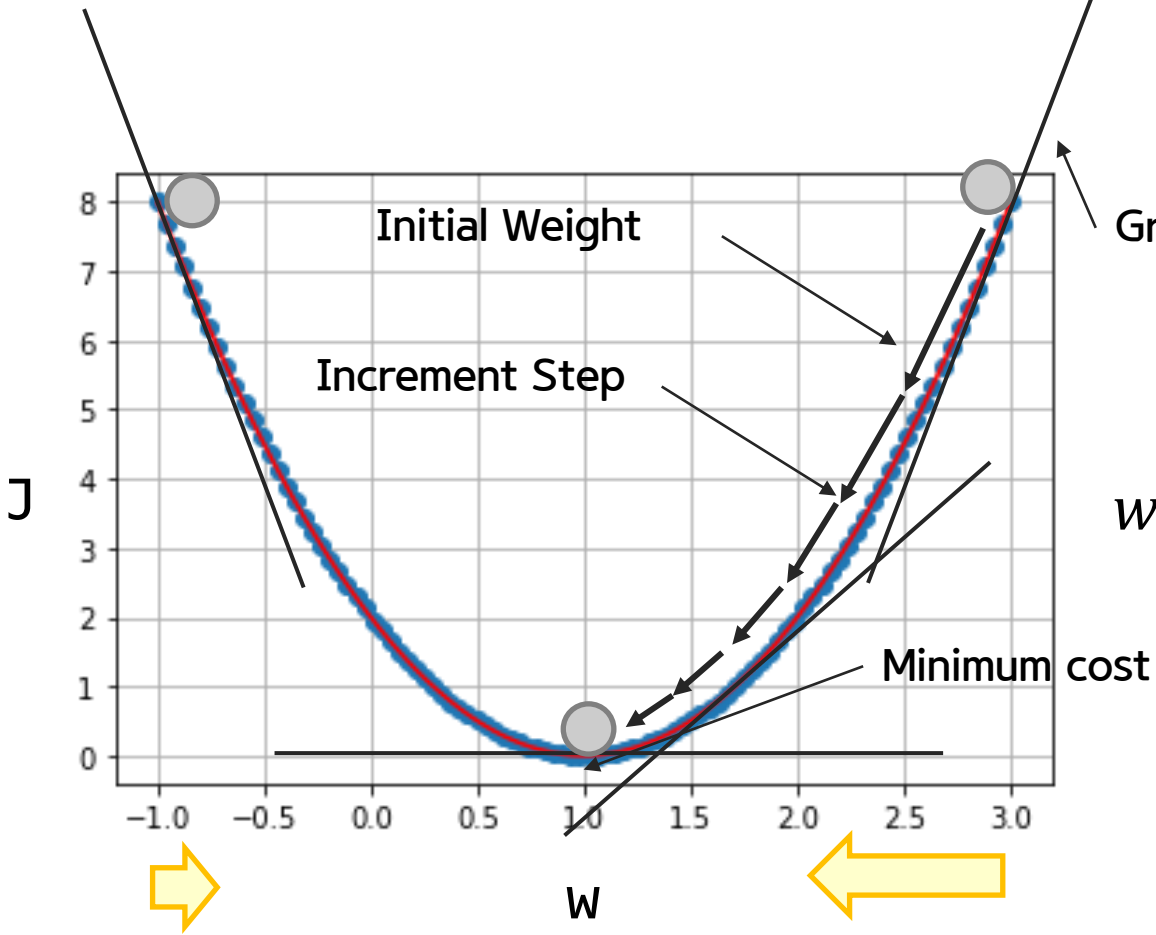
규제전 : -0.08
규제후 : -0.07

$x = 2$ $\quad l_1 = 0.01$

$y = 2$

$\alpha = 0.01$

Gradient descent algorithm



$$\hat{y} = wx + b$$

$$w = w - \alpha(\hat{y} - y)\mathrm{x}$$

0.08

$$w = w - \alpha(\hat{y} - y)\mathrm{x} + 0.01 * \mathrm{sign(w)}$$

+0.08 - 0.01

규제전 : +0.08
규제후 : +0.07

$x = 2$ $\quad l_1 = 0.01$

$y = 2$

$\alpha = 0.01$

L1 규제는 손실 함수에 가중치의 절대값인 L1 노름(norm)을 추가한다.

$$\frac{\partial}{\partial w}L = (a-y)x + \alpha * sign(w)$$

$$w = w - \eta\frac{\partial L}{\partial w} = w - \eta(a-y)x + \alpha * sign(w)$$

파이썬으로 작성된 L1 규제 적용된 오차 역전파 구현

```
w_grad += alpha * np.sign(w)
```

회귀 모델에 L1 규제를 추가한 것을 라쏘 모델이라 한다.

L2 규제는 손실 함수에 가중치에 대한 L2 노름(norm)의 제곱을 더한다.

$$\left|\left|w\right|\right|_2 = \sum_{i=1}^{n} |w_i|^2$$

$$L = -(ylog(a) + (1-y)\log(1-a))$$

L1 노름을 그냥 더하지 않고 규제의 양을 조절하는 파라미터 α 를 곱한 후 더한다.

$$L = -(ylog(a) + (1-y)\log(1-a)) + \frac{1}{2}\alpha \sum_{i=1}^{n} |w_i|^2$$

L2 규제는 손실 함수에 가중치에 대한 L2 노름(norm)의 제곱을 더한다.

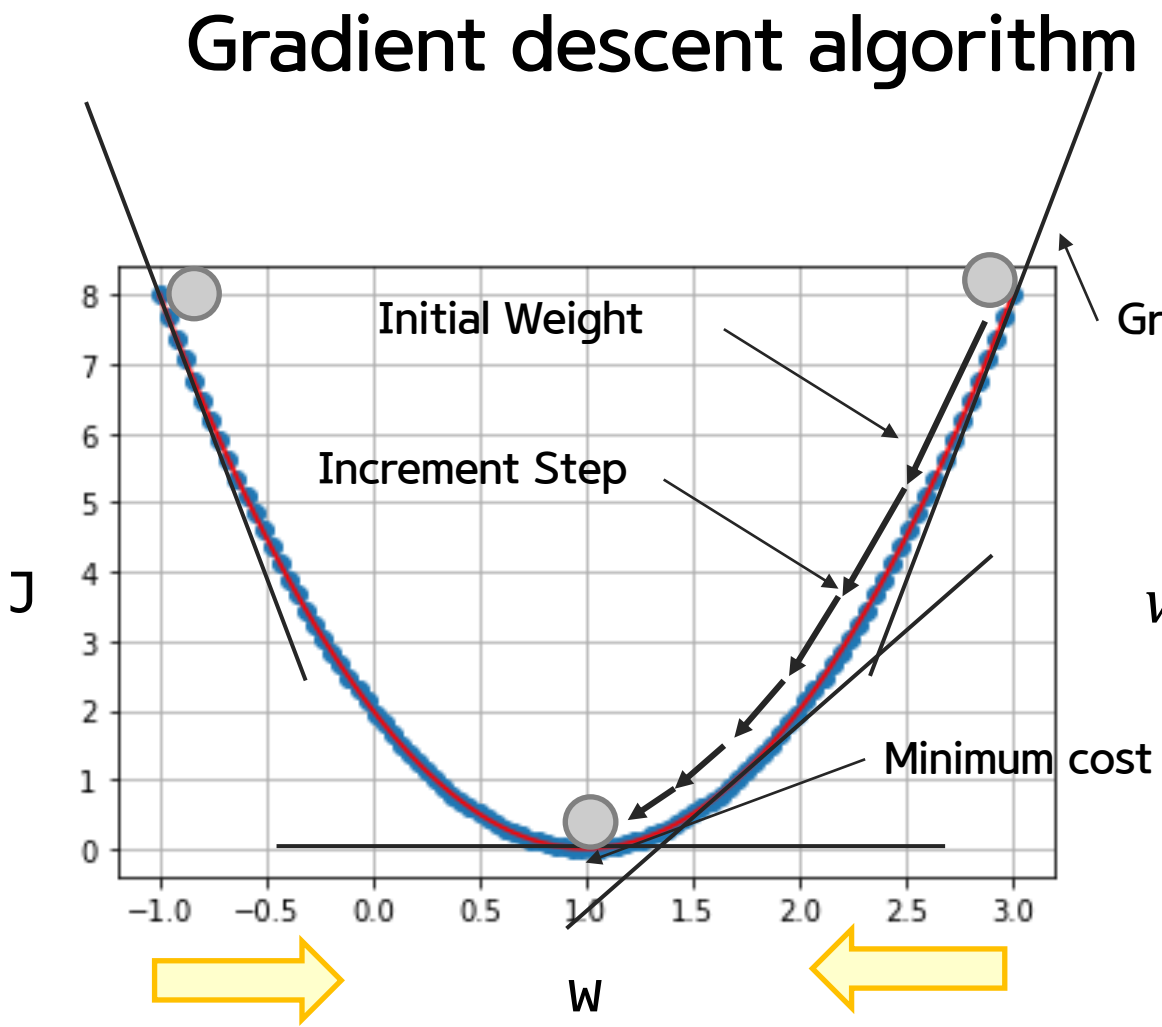
$$\frac{\partial}{\partial w} L = -(y - a)x + \alpha * w$$

$$w = w - \eta \frac{\partial L}{\partial w} = w - \eta(a - y)x + \alpha * w$$

파이썬으로 작성된 L2 규제 적용된 오차 역전파 구현

```
w_grad += alpha * w
```

회귀 모델에 L2 규제를 추가한 것을 릿지 모델이라 한다.



$$\hat{y} = wx + b$$

$$w = w - \alpha(\hat{y} - y)\mathrm{x}$$

0.08

$$w = w - \alpha(\hat{y} - y)\mathrm{x} + 0.01 * \mathrm{w}$$

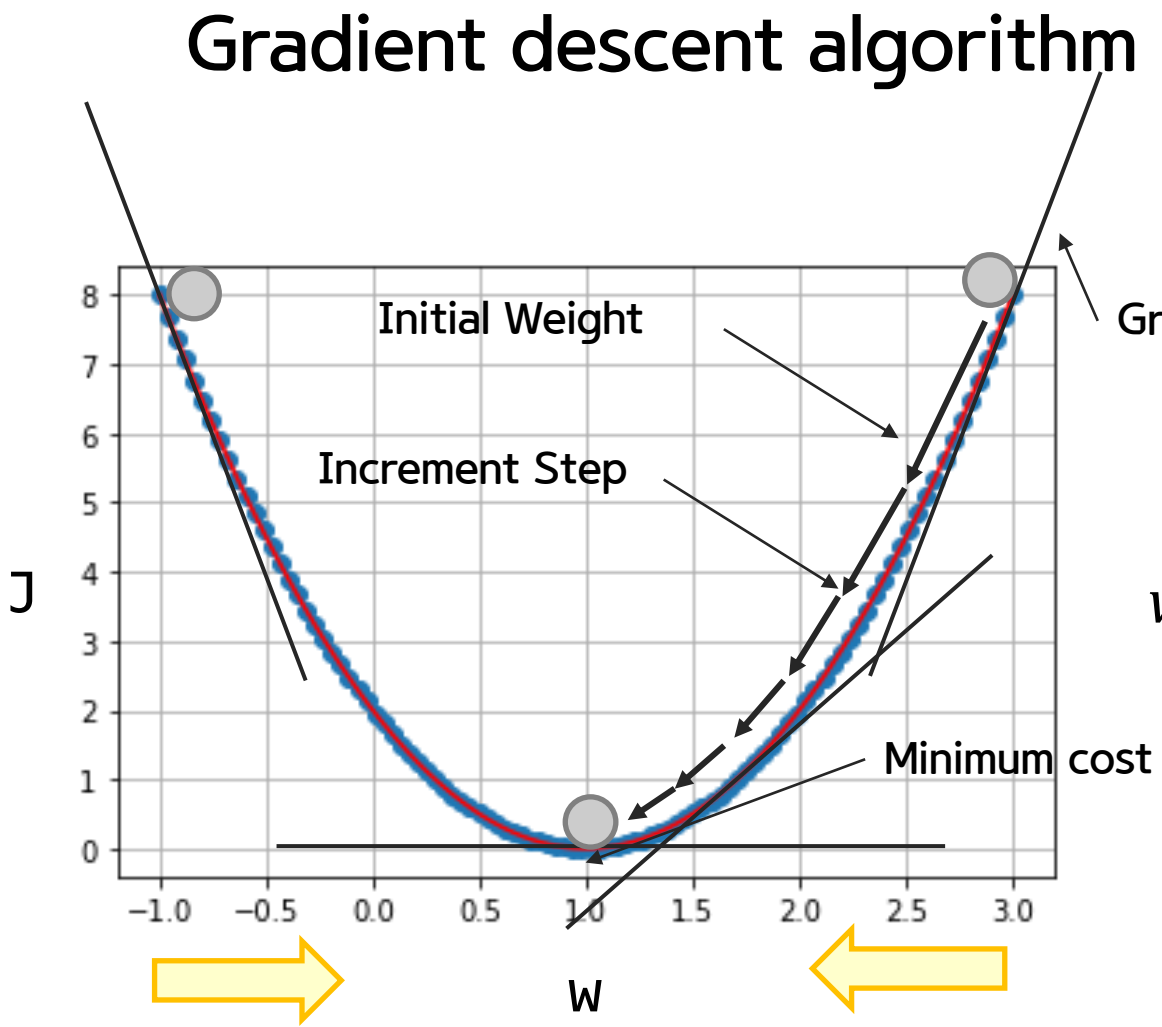
-0.08 + 0.03

$x = 2$

$y = 2$

규제 전 : -0.08

규제 후 : -0.05



$$\hat{y} = wx + b$$

$$w = w - \alpha(\hat{y} - y)\mathrm{x}$$

0.08

$$w = w - \alpha(\hat{y} - y)\mathrm{x} + 0.01 * \mathrm{w}$$

+0.08 - 0.01

$x = 2$

$y = 2$

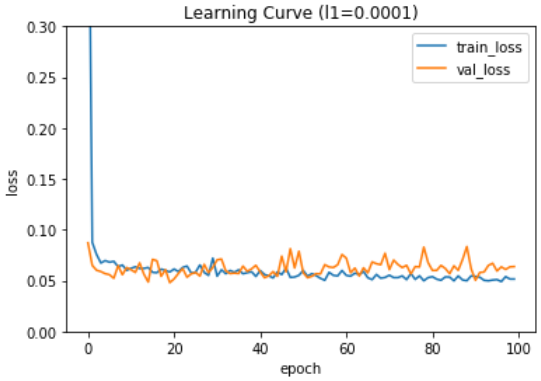
규제 전 : +0.08

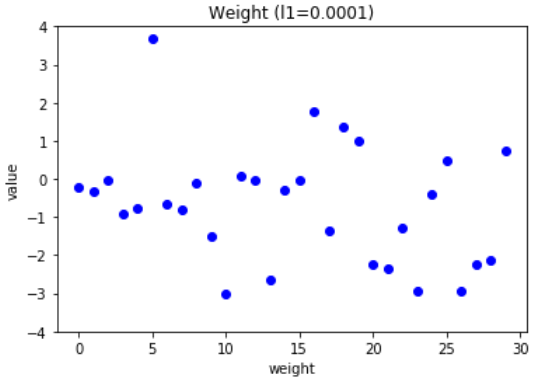
규제 후 : +0.07

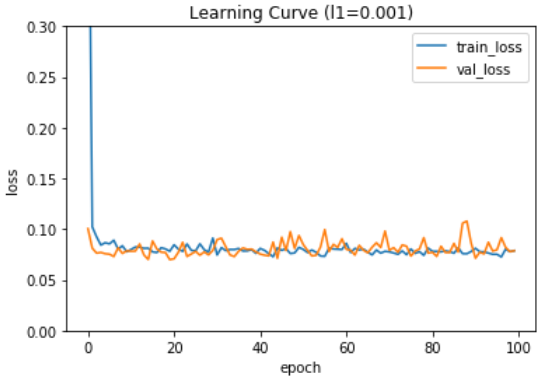
L1 규제 적용

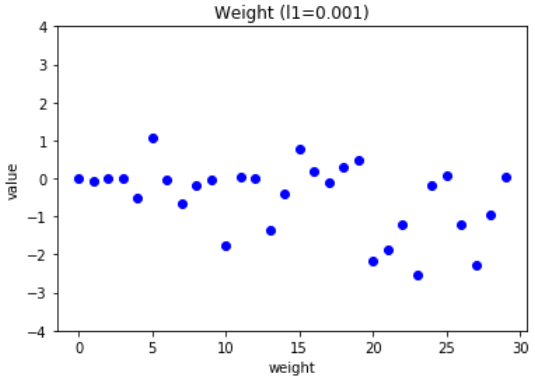
```
l1_list = [0.0001, 0.001, 0.01]

for l1 in l1_list:
    lyr = SingleLayer(l1=l1)
    lyr.fit(x_train_scaled, y_train, x_val=x_val_scaled, y_val=y_val)

    plt.plot(lyr.losses)
    plt.plot(lyr.val_losses)
    plt.title('Learning Curve (l1={})'.format(l1))
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train_loss', 'val_loss'])
    plt.ylim(0, 0.3)
    plt.show()

    plt.plot(lyr.w, 'bo')
    plt.title('Weight (l1={})'.format(l1))
    plt.ylabel('value')
    plt.xlabel('weight')
    plt.ylim(-4, 4)
    plt.show()
```

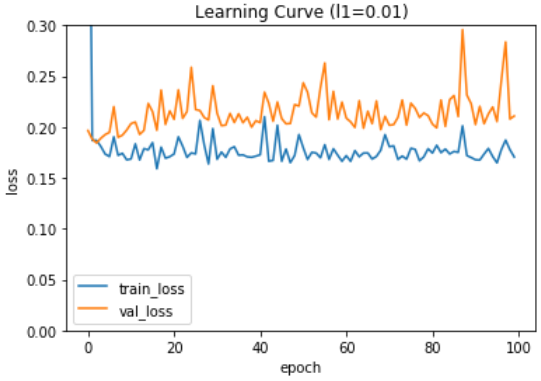
L1 규제 적용

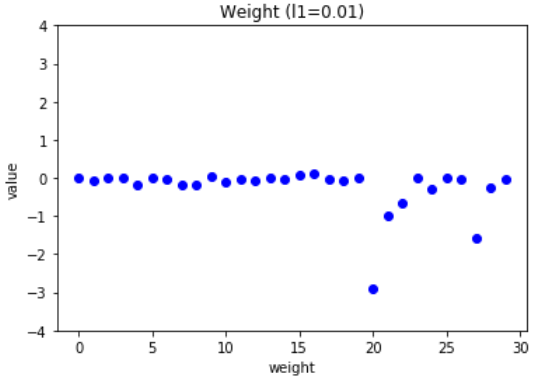

Learning Curve (l1=0.0001)
train_loss
val_loss
loss
epoch


Weight (l1=0.0001)
value
weight


Learning Curve (l1=0.001)
train_loss
val_loss
loss
epoch


Weight (l1=0.001)
value
weight

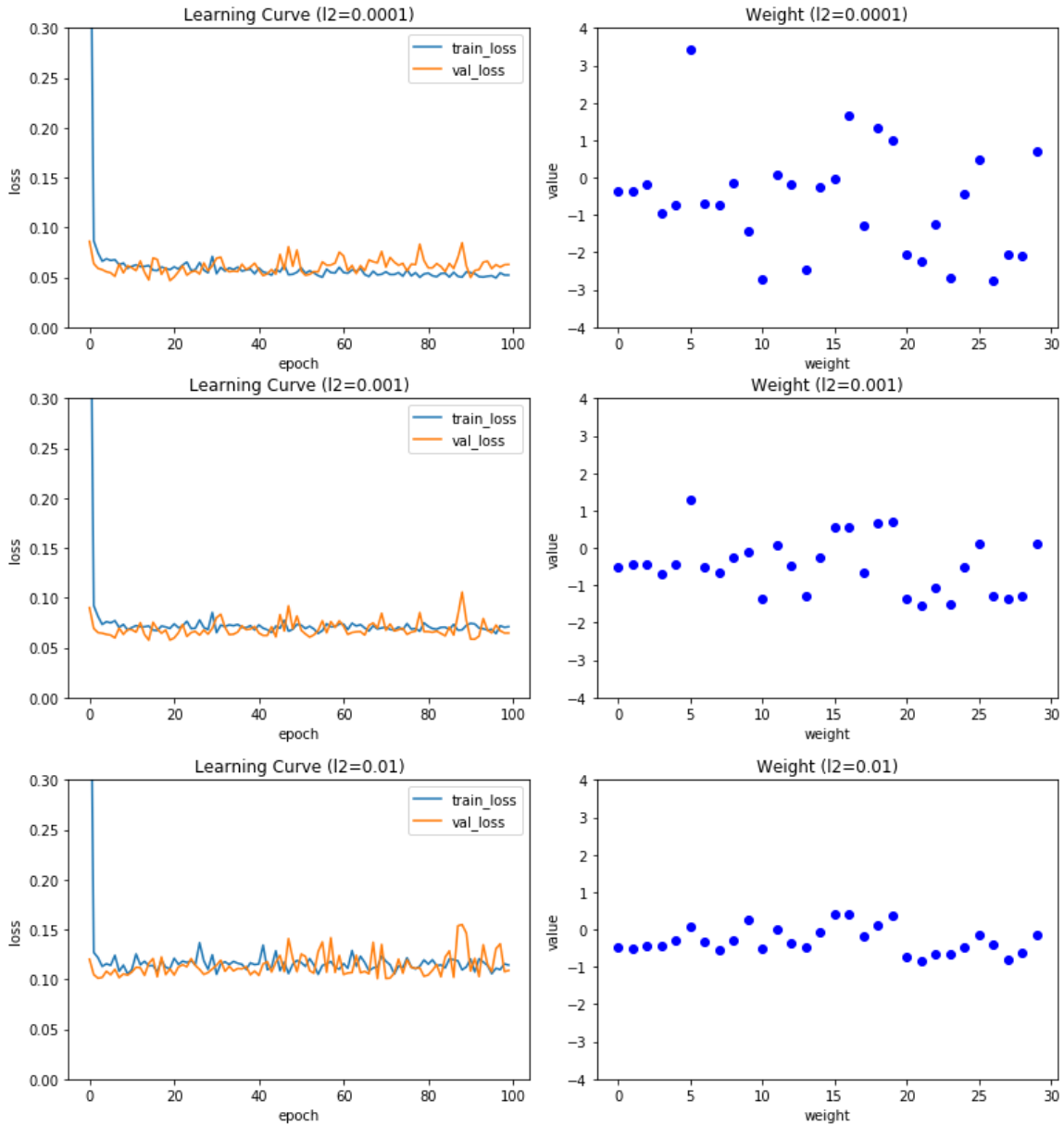

Learning Curve (l1=0.01)
train_loss
val_loss
loss
epoch


Weight (l1=0.01)
value
weight

L2 규제 적용

```
l2_list = [0.0001, 0.001, 0.01]

for l2 in l2_list:
    lyr = SingleLayer(l2=l2)
    lyr.fit(x_train_scaled, y_train, x_val=x_val_scaled, y_val=y_val)

    plt.plot(lyr.losses)
    plt.plot(lyr.val_losses)
    plt.title('Learning Curve (l2={})'.format(l2))
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train_loss', 'val_loss'])
    plt.ylim(0, 0.3)
    plt.show()

    plt.plot(lyr.w, 'bo')
    plt.title('Weight (l2={})'.format(l2))
    plt.ylabel('value')
    plt.xlabel('weight')
    plt.ylim(-4, 4)
    plt.show()
```

L2 규제 적용


Learning Curve (l2=0.0001)
train_loss
val_loss
loss
epoch
Weight (l2=0.0001)
value
weight
Learning Curve (l2=0.001)
train_loss
val_loss
loss
epoch
Weight (l2=0.001)
value
weight
Learning Curve (l2=0.01)
train_loss
val_loss
loss
epoch
Weight (l2=0.01)
value
weight

3. 신경망 시작하기

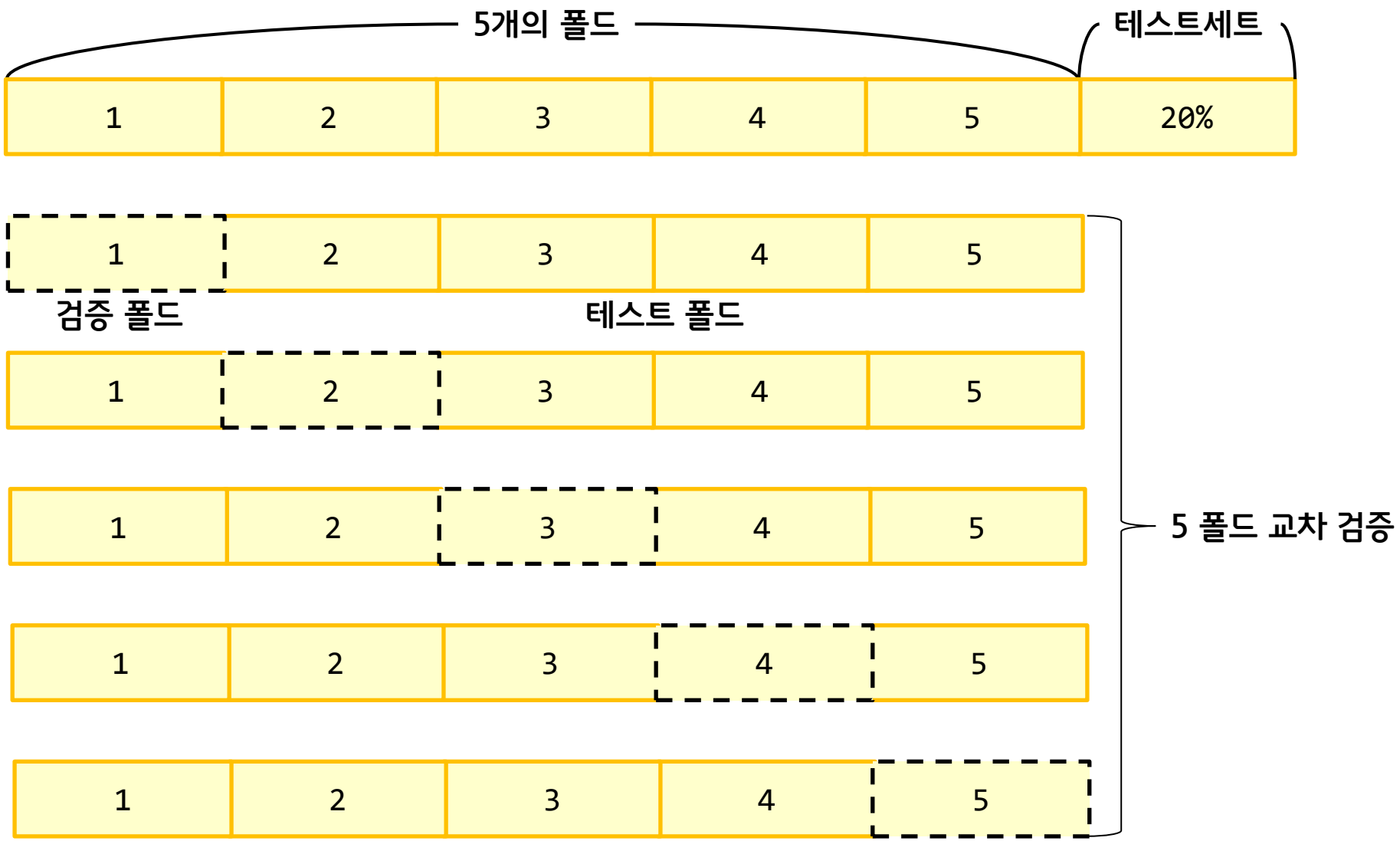
3.1 단일층 신경망 구현

3.2 검증 세트 분리와 스케일링

3.3 과대적합과 과소적합

3.4 규제방법 구현

3.5 교차 검증 구현

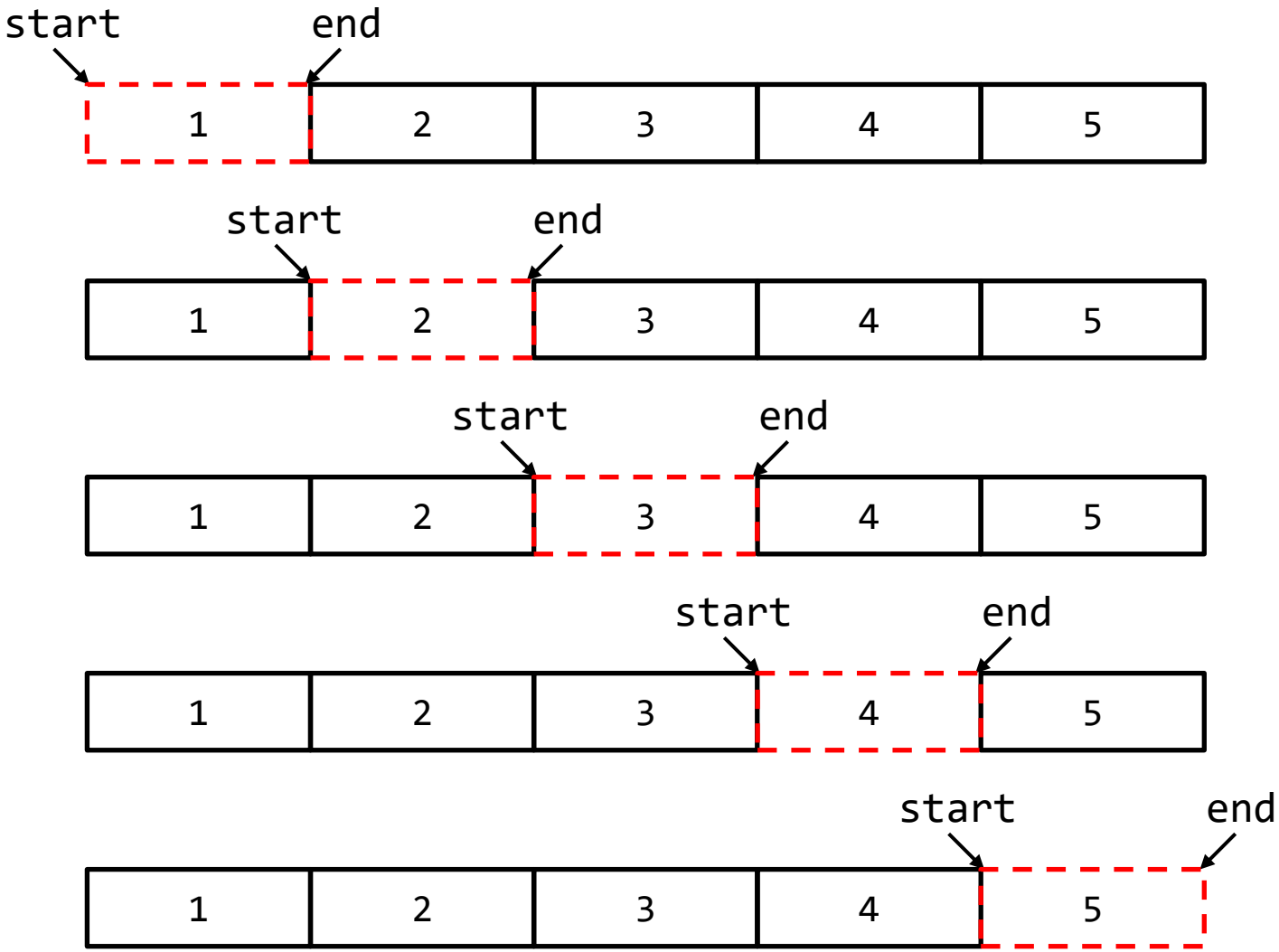

5개의 폴드
테스트세트
1
2
3
4
5
20%
검증 폴드
테스트 폴드
5 폴드 교차 검증

교차 검증 과정

1. 훈련 세트를 k개의 폴드(fold)로 나눈다.
2. 첫 번째 폴드를 검증 세트로 사용하고 나머지 폴드(k-1개)를 훈련 세트로 사용 한다.
3. 모델을 훈련한 다음에 검증 세트로 평가 한다.
4. 차례대로 다음 폴드를 검증 세트로 사용하여 반복한다.
5. k개의 검증 세트로 k번 성능을 평가한 후 계산된 성능의 평균을 내어 최종 성능을 계산한다.

```
validation_scores = []
k = 10
bins = len(x_train_all) // k

for i in range(k):
    start = i*bins
    end = (i+1)*bins
    val_fold = x_train_all[start:end]
    val_target = y_train_all[start:end]

    train_index = list(range(0, start))+list(range(end, len(x_train)))
    train_fold = x_train_all[train_index]
    train_target = y_train_all[train_index]
    train_mean = np.mean(train_fold, axis=0)
    train_std = np.std(train_fold, axis=0)
    train_fold_scaled = (train_fold - train_mean) / train_std
    val_fold_scaled = (val_fold - train_mean) / train_std

    lyr = SingleLayer(l2=0.01)
    lyr.fit(train_fold_scaled, train_target, epochs=50)
    score = lyr.score(val_fold_scaled, val_target)
    validation_scores.append(score)

print(np.mean(validation_scores))
```

0.9711111111111113


start
end
1
2
3
4
5
start
end
1
2
3
4
5
start
end
1
2
3
4
5
start
end
1
2
3
4
5
start
end
1
2
3
4
5